PERSEUS, Episode V - The Avionics strikes back: A highly adaptive, COTS-based and value-driven avionics for PERSEUS small reusable launcher

*Mickaël GOMES *, Sophie MISSONNIER* and David TCHOU-KIEN ***
** CT Ingénierie*
Immeuble Arago 1, 41 bvd Vauban, 78280 Guyancourt, France
mickael.gomes@ctingenierie.com
sophie.missonnier@ctingenierie.com
*** Centre national d'études spatiales (CNES)*
2 Place Maurice Quentin, 75001 Paris, France
david.tchoukien@cnes.fr

Abstract

PERSEUS program from CNES has been offering students the opportunity to contribute to the design, manufacture, testing and launch of sounding rockets for 17 years.
These conditions, unlike traditional development of space systems, require work steps to be quick (students are often involved for only a few months) and light (in learning and economic resources), making cost and openness design criteria; prioritization of needs becomes key to ensure continuous progress and delivery on time for launch campaigns.
PERSEUS follows Agile principles to incrementally develop a set of modular, flexible and adaptive building blocks including avionics systems described in this article, for ASTREOS family successive bi-liquid launchers.

1. Specificities of the context and adapted development method

1.1 PERSEUS project background

The space sector is undergoing a major shift, with disruptive innovations and new players. For the last 17 years, the PERSEUS program [1] of the French space agency (CNES) is a cornerstone of the relationship between future engineers, academic institutions and the space industry. Indeed PERSEUS aims at raising students' awareness of these economic, strategic and scientific dimensions, promoting the research of promising technologies and developing vocation for the "NewSpace".

For the past few years, many space-eager student associations have the opportunity to design experimental rockets of modest size based on low-power solid propulsion, and launch them during dedicated annual events such as the European Rocketry Challenge (EuRoc) or the C'Space. These events are an interesting entry point into the space field, especially to understand the basics of flight dynamics, but they do not put emphasis on the constraints and complexity of real modern launchers such as reusability with sophisticated GNC capabilities or high-pressure liquid cryogenic propulsion system.
Therefore, we involve hundreds of college students dispatched throughout France in the design and manufacture of a small reusable bi-liquid LOX-Ethanol propelled launcher named ASTREOS, which is launched from Esrange Space Center in Northern Sweden alongside industrial and commercial rockets.

For that purpose, PERSEUS relies on contributions and inputs from dozens of universities, schools and associations; each student can be involved for short periods of time (a few hours a week for a few months) or for longer periods of time (several years in their curriculum, internship or gap year and in an associative framework).
As far as avionics is concerned (but the same applies to other areas), each entity has its own teaching, e.g. preferred computing environment or modeling framework (such as SolidWorks / CATIA, Matlab / LabView) and its specialties

depending on whether they are more electronics oriented, software programming focused, or rather generic and versatile. Consequently, mind-set and skills of students from a given school often prove to be more suitable for certain tasks than others.
Despite these differences in school habits and work environments, one of the pitfalls to be avoided is that everyone starts from scratch each time, so capitalizing on developments and smoothly transferring modules from teams to teams is a major challenge for the team coordinating the PERSEUS student work.

1.2 Avionics development strategy

Thus, a good way to standardize and facilitate software development over time, across all these diverse participants is to use proprietary framework and software as little as possible because it would require too much effort (in time, money and learning) to adapt the produced blocks from one environment to another. In a similar way for hardware platforms, it seems unwise to choose a proprietary board linked to this or that environment, e.g. CompactRIO for National Instruments or Speedgoat for Simulink Real-Time™.
In order to achieve an effective collaboration between our partners and a straightforward integration of the produced modules, while keeping financial investments as low as possible, the best strategy is the extensive use of open-source software and generic mainstream hardware platform.
This strategy of openness allows to lower costs, and at the same time to avoid:

- to compartmentalize and get stuck in an environment or to be dependent on a provider
- to require from schools to make an investment prior to their participation in PERSEUS, which results into getting a very wide range of participating schools

On the hardware side, homemade microcontroller-based systems show a clear lack of simplicity and scalability, implying not only software but also hardware adjustments in order to fulfil each functional or performance evolution; the sum of specification, hardware and software development, integration and test leads to a process that is far too burdensome for the objectives and context of PERSEUS.
For reasons of:

- modularity with numerous flexible and evolving interfaces
- simplicity of implementation as SSH/FTP client is sufficient for a complete remote development environment
- portability as produced code and library are not platform-dependent
- scalability notably for evolved and resource-hungry functionalities like GNC, AFTS or video stream
- ease of maintenance,

… a Single Board Computer (SBC) is used rather than a microcontroller.
After a thorough comparison of available mainstream SBC based on various versatility and performance criteria, taking into account that it is the most widely studied in engineering courses and also one of the least expensive, we have chosen the Raspberry Pi 4 as a building block for the avionics development. It is based on a 64-bit quad-core Cortex-A72 processor with up to 8GB RAM, hardware-based 1080p30 encode capability and Gigabit Ethernet, while keeping a maximum power consumption of 15W. Nonetheless, to optimize occupation of a very restricted volume on-board and to minimize energy consumption, we also use the Raspberry Pi 0 with its tiny size (65 × 31 × 13 mm) and reduced performance for less demanding systems like sensor data acquisition modules.

On the software side, for comparable reasons of:

- cost without mandatory proprietary licenses to pay,
- performance as it remains the most efficient language,
- versatility despite its low-level character,
- support in schools and still considered as a key element in embedded software engineering,
- large availability of software libraries,

… the C language and open-source libraries are used as much as possible rather than proprietary ones.
One limitation of this logic lies in the GNC side, since Matlab/Simulink keeps being the most efficient means in PERSEUS context to quickly design and adapt the control laws, to simulate the mission. However this use of a commercial platform is limited to the strict minimum; once designed, output laws and code are then ported to C.

1.3 Agile methodology

Given the objectives of PERSEUS to move quickly from one demonstration step to the next one, and its development team constituted of various actors, mainly successive students from different courses involved on a limited time and remote locations, the development process needs to be broken down into small blocks that are successively assembled and integrated until the rocket is flight-ready.

Such scheme is clearly in line with the Agile approach that has emerged few decades ago to adopt a more dynamic and collaborative methodology for software developments, resulting in several lightweight development methodologies like Extreme Programming (XP), Scrum and Test-Driven Development (TDD).
The Agile methodology [2] is based on emphasizing the importance of:

- **interactions** over process,
- **working software** over documentation,
- **collaboration** over negotiation,
- **responding to change** over following a plan.

All these factors stay important and useful in building software solutions, but keeping in mind this "new" order of priority will help towards better satisfying customers.
More widely than for software development, the agile approach of greater focus on people, a working solution (to be enriched incrementally), collaboration (with client), and flexibility (with regards to requirements), enables many types of teams to adopt a mind-set giving more ability to answer new challenges.
In accordance with the Agile principles, the PERSEUS team follows an iterative and incremental workflow for the whole development of the space launch demonstrators and rockets, inspired from the Scrum framework; much like a rugby team (where it gets its name) training for the big game, Scrum encourages teams to learn through experiences, self-organize while working on a problem, and reflect on their wins and losses to continuously improve. This method is applied also at the level of the different domains: obviously avionics, GNC, Ground segment, etc. and their associated products.

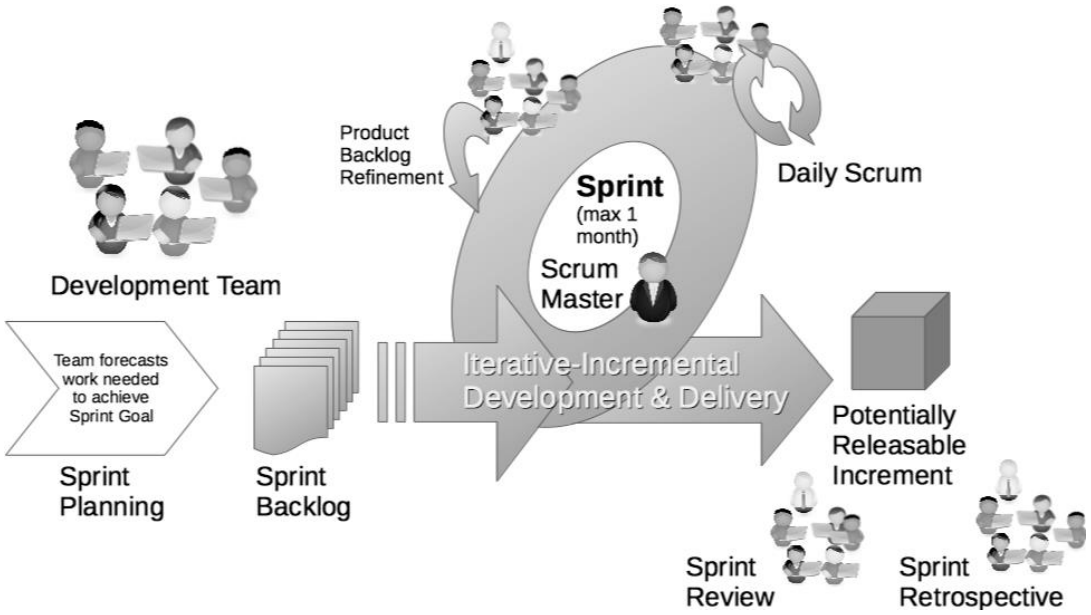

Figure 1: Scrum iterative and incremental workflow [3]

Traditional Scrum workflow as shown in Fig.1 is adapted and lightened, e.g. since student teams often have only one or half a day per week to contribute it is impossible to hold a daily meeting, moreover the role of Scrum Master and Product owner are actually assumed by the same person, namely the PERSEUS coordinator of a specific domain (avionics, ground segment, …).
However, we keep the pace of a Sprint every 2 to 4 weeks with a unique remote meeting for each team, which serves as a mixed all-in-one Sprint planning in order to agree on the next Sprint Goals and on the expected deliverables, review produced outputs and discuss encountered difficulties through a retrospective. This 20 to 30 minutes meeting every 2-4 weeks appears to be the best balanced between autonomy and responsibility given to students, technical support that is essential to help them move forward and overcome certain obstacles and time spent to ensure effective flow of activities spread over dozens of student groups; and all this thanks to the structure provided by the Scrum method.
Thus PERSEUS avionics is developed through continuous integration of a set of modular, flexible and adaptive building blocks, designed and incrementally enhanced to rapidly evolve across ASTREOS successive iterations; ASTREOS version's capabilities are allocated to avionics products, and the relevant universities are involved with targets and regular deliverables, starting with a prototype that meets the basic operational needs which is increased in complexity as development goes along.

2. A ruggedized and functionalities-enhanced mainstream Single Board Computer (SBC)

As previously stated, for versatility, performance and cost reasons, we have made a thorough comparison of available mainstream generic Single Board Computer (SBC) based on various comparison criteria, e.g. supported operating system, degree of portability of homemade software and hardware extension from one SBC to the other, available standardized and/or open-source libraries and how widespread the board is in engineering schools. Raspberry Pi 4 and Zero have appeared to be the best candidates, with the advantage of easily reusing software bricks notably GPIOs-related, through two form factors meeting distinct needs, i.e. computationally demanding operations for complex systems like the On-Board Computer and lower space and power consumption for distributed measurement and control systems.

Nonetheless, such mainstream products lack some characteristics to be operational, ruggedized and flight-ready systems compatible with the flight dynamics and environment. First of all, electromagnetic interference (EMI) must not be neglected, they are induced by sources both inside the rocket through the stacking of several electronic equipment, and outside the launcher from radio-frequency equipment and a harsh very cold and humid environment in the Swedish Space Center (SSC), which is our usual launch base. Furthermore, despite a native support of many protocols thanks to the GPIO pins, like SPI, PWM or basic discrete signals, we need an even broader support of both digital and analog communication protocols in order to integrate a large variety of data measurement and actuator control systems.

The best efficiency-driven solution is then to conceive a generic extension board to enhance these mainstream hardware platforms through a minimalist design, and turn them into a ruggedized fully featured low-cost flight-ready system.
A first result of such an approach is shown in Fig.2, with a Raspberry Pi 4 directly coupled to a homemade extension board in order to manufacture the On-Board Computer of PERSEUS latest solid sounding rocket, named SERA IV. One can observe that a particular attention has been given to reduce EMI. A greater electromagnetic compatibility level is achieved by decoupling and applying a systematic galvanic isolation on each on-board system through key component such as:

- DC-to-DC converter to isolate and regulate internal power circuit from common backbone power circuit, with an example of a sizeable Traco Power THN 20-2411 on the top side
- Optocoupler to transfer digital signals between two isolated circuits, notably three Isocom IS281 4-channel transistor on the bottom side
- A clear and large separation of ground planes on the top side of the PCB in order to break potential ground loops, prevent crosstalk and globally reduce electrical noise

Each reference has obviously been chosen after an exhaustive analysis of available components on the market in accordance with distinct comparison criteria such as input voltage range, power efficiency, electromagnetic compatibility, channel capacity, available bandwidth, occupied space, and, last but not least, price.
To ensure the compatibility of on-board systems with the vibration and acceleration levels that they have to undergo, usual data interfaces like RJ45 or USB connectors can't be used, therefore they have been unsoldered on the Raspberry Pi 4, and these interfaces have been redirected through a tougher D-Sub connector and the intermediate EMI reducer stage. But as Ethernet magnetics are no longer presents, an additional transformer with a schematic identical to the removed connector has been added on the bottom side via an Abracon ALANS10001.
Such unsoldering process is quite long and tedious, and could be avoided with a SBC offering only one generic pluggable connector, and it's precisely what the Raspberry Pi Foundation offers through the Compute Module 4 with a smaller form factor and a couple of 100-pin high-density Hirose connectors on the underside, nonetheless vibration tests must be carried out to verify that mating connectors are compatible with flight environments.

Regarding the need to enable a wider support of digital communication protocols, a straightforward approach is to integrate a COTS module with minimal design effort through a dedicated hole on the PCB, instead of designing a specific routing with appropriate passive components in order to operate it as any other integrated circuit (IC). This approach has been used to rapidly add an FTDI USB-RS485 PCB adapter instead of a dedicated circuit for the FT232RQ IC with all necessary electronics, as RS485 communication protocol was a minor need only used in pre-flight operations to configure our inertial measurement system and will not be used in upcoming iterations of ASTREOS. A global COTS-based electronic design can then help to put in practice Agile methodology and improve our ability to adapt and quickly respond to new needs that can emerge from other domains like ground segment, GNC or propulsion system.

Nonetheless, it is more difficult to apply this approach to analog circuits since a specific signal conditioning must be carried out and an optimal filtering operated to maximize the signal-to-noise ratio for each sensor, as some will require signal amplification to correctly process tens of millivolts while other will need a voltage divider to lower tens of volts. Moreover a particular attention must be brought to the integration of analog circuits in order to split analog and digital sections and keep a high level of electromagnetic compatibility, notably through a thin trace between each ground planes. A certain degree of versatility and reusability of a dedicated analog circuit can however be achieved if a synthesis of all probable input signals is conducted and taken into account to choose key components like an Analog-to-Digital converter with a large bandwidth, programmable sampling rate and a sufficient precision to compute each sensor data.

Finally, it is important to notice that such extension board is usually not needed when a student team contributes to PERSEUS avionics and conceive a complex software system like the OBC, as ruggedize-oriented modifications are only necessary for a flight-ready system and COTS modules can be bought in a fully Plug-and-Play boxed format rather than raw to solder format. Henceforth, students are usually completely unaware of this necessary upgrades, and they only need one or a few Raspberry Pi and some COTS related to the system under development. This strategy not only facilitates the design, validation and integration processes, but also brings an obvious gain in terms of financial participation and project takeover effort by the dozen of teams building ASTREOS launcher.

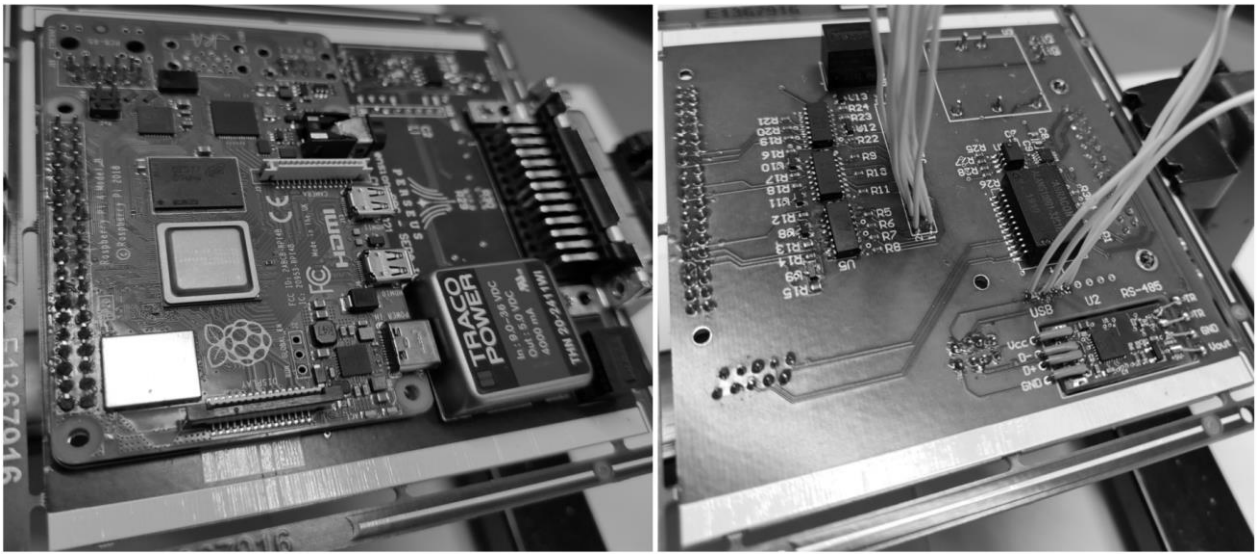

Figure 2: Top (left) and bottom (right) side of our extension board

3. A critical Ethernet-based communication network

3.1 Real Time Ethernet overview

In order to ensure scalability, flexibility and versatility of ASTREOS-1's backbone communication network, the old paradigm based on cohabitation of several generic and fieldbus protocols (USB, RS422, CAN…) is clearly outdated, and can't respond to growing data transmission needs implied by sophisticated GNC capabilities or even video streaming.
Real Time Ethernet (RTE) network is an effective answer seriously considered by the academic field for almost two decades, leading to comprehensive comparisons [4][5] of common industrial products like PROFINET, EtherCAT, Ethernet POWERLINK (EPL), SERCOS III, Modbus, and lesser spread products such as JetSync, PowerDNA, SynqNet, ControlNet or TCnet.
For all these products, the main challenge is to ensure a time-determinist communication with real-time guarantees like minimal transfer delay, response time and jitter, while focusing on precise periodic exchange of small data records through a temporally and bandwidth constrained traffic. Automotive, factory automation or more recently launch vehicles [6][7] are some of the sectors where RTE network brings a clear breakthrough compared to traditional communication systems.
Depending on the field of application, some characteristics of RTE technologies are prioritized over others, for example delivery time, number of end nodes, network topology or the possibility of mixed RTE and non-RTE bandwidth may

be put forward by some [8], while others [9] emphasize redundancy capacity, minimal redundancy recovery time or time synchronization techniques and accuracy between end-stations.
Several performance indicators are designed to pragmatically compare and evaluate each product [10][11], for instance the minimum cycle time (MCT) a system can achieve based on a specific number of nodes and frame size, varies from ten microseconds to hundreds of milliseconds.

The most common classification of RTE protocols is based on the integration level compared to UDP/TCP/IP and Ethernet layers, as shown in Fig.3. A common base for all Ethernet network is the two or four twisted-pair cables universally used as physical layer. Henceforth, non-real-time applications make use of typical Internet protocols, standard IP stack and Ethernet layer as defined in ISO 8802-3. The first RTE approach is to keep the TCP/UDP/IP protocols unchanged and concentrate all real-time modification in the top layer; this solution is called “on top of TCP/IP”. In the second approach, the TCP/UDP/IP protocols are bypassed and the Ethernet layer is accessed directly (“on top of Ethernet”). In the third approach, the Ethernet mechanism and infrastructure itself is modified to make it more real-time performed (Modified Ethernet). Of course, depending on the chosen approach, the minimum cyclic time will differ greatly, standard hardware and nodes will or will not be compatible, and the random nature of Ethernet protocol will be more or less overcome.

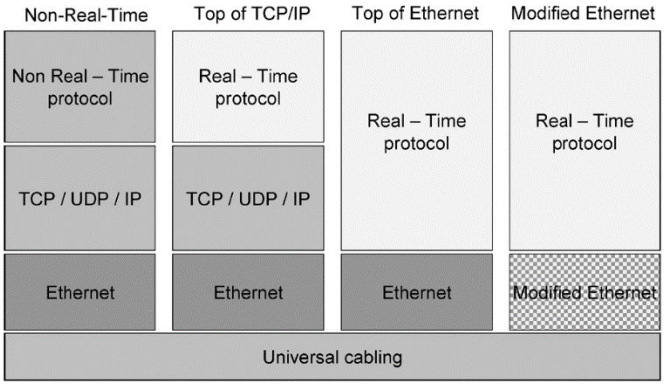

Figure 3: RTE categorization [5]

PROFINET or Modbus protocols are based on the first approach, and benefit from an easier integration with standard IP stack, but can’t reach less than 5 ms of MCT with a hardly deterministic and predictable behavior. Real-time guarantees are truly observed through second method with MCT from 100µs to 1ms and the possibility to use ordinary Ethernet hardware, and even mix IP-based on non-IP nodes, but with a significant impact on the time-determinist subpart. The last approach used by SERCOS III or EtherCAT is obviously the most performant with MCT lower than 100µs but specific hardware must be used and can’t coexist with regular Ethernet nodes.

In line with the overall avionics development strategy, the best fitted protocol to meet our needs must be open-source oriented, with dependencies to any provider as limited as possible, and compatible with Raspberry Pi (RPi) generic hardware platform. Consequently, protocols belonging to the third approach cannot be used. On the other hand, protocols belonging to the second category offer a good balance between flexibility, versatility and real time performance, notably through a well-known protocol named Ethernet POWERLINK (EPL). It is based on the principle of a master–slave scheduling system with a polling scheme. The master, called the managing node (MN), controls all accesses to the medium, ensures real-time access to the cyclic data and lets non-real-time IP frames pass through only in time slots reserved for this purpose. The polling order is preconfigured before runtime but may be modified during execution. All other nodes are called controlled nodes (CNs) and are only allowed to send on request by the MN. The MN sends a multicast start-of-cycle (SoC) frame to signal the beginning of a cycle. The send and receive time of this frame is the basis for the common timing of all the nodes. It is important to keep the start time of an EPL cycle as exact (jitter-free) as possible. As shown in Fig.4, the following periods exist within one cycle: Start period, Isochronous period, Asynchronous period, and an additional Idle period. The length of an individual period can vary within the preset period of an EPL cycle.

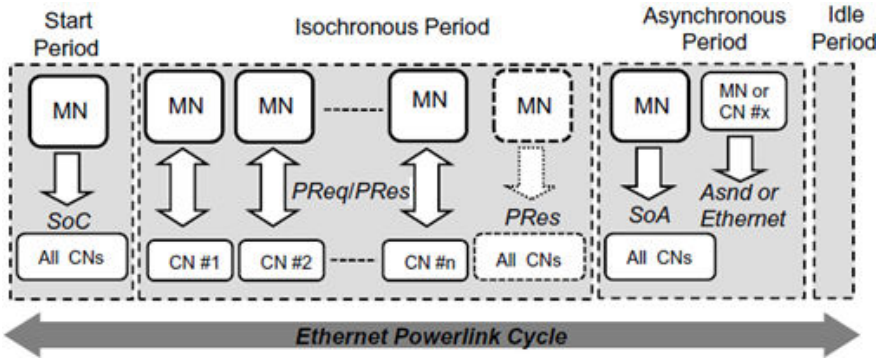

Figure 4: EPL cycle and sub-periods [11]

In the Isochronous period of the cycle, a Poll-Request (PReq) unicast frame is sent consecutively to every configured and active node. The accessed node responds with a multicast Poll-Response (Pres) frame where it may indicate that it has asynchronous traffic to send. The responses are sent in broadcast; traffic from node to node is hence supported. Depending on the additional traffic requests, the master will start a sporadic window named Asynchronous period where EPL network access may be granted to one CN or to the MN based on a priority scheduling policy, for the transfer of a single asynchronous message only. The preferred protocol for asynchronous messages is UDP/IP. The start-of-asynchronous (SoA) is a confirmation sent to CNs that all isochronous data has been exchanged during the isochronous period. Thus, transmission of isochronous and asynchronous data will never interfere, as EPL offers hard guarantees for the preplanned traffic and soft guarantees regarding the on-demand traffic.

3.2 openPOWERLINK over PREEMPT-RT on Raspberry Pi

Due to its open protocol nature, EPL communication can easily be designed and integrated into a hardware-based solution (FPGA, ASIC, …) for performance purpose, it is in fact the usual choice in the industry, but it is less fitted to our application case, as it would require to bypass Raspberry Pi's Ethernet chipset through an additional dedicated network card. Fortunately, an open-source software-based implementation is also available via openPOWERLINK (OPL) [12], managed by the Ethernet POWERLINK Standardization Group (EPSG), and supporting all major EPL features such as Standard, Multiplexed and Poll_Response Chaining mode of operation, MN redundancy, SDO via ASnd and UDP, as well as asynchronous communication via a Virtual Ethernet interface.
A two layer architecture is used by OPL with a user layer containing the application-centric code such as the object dictionary while the kernel layer contains the time critical parts (e.g. data link layer). The user layer is composed of an API that can be used by any application to configure and operate an OPL communication, and a set of software modules linked to OPL stack that are bridged to other kernel specific modules through a communication abstraction layer (CAL). The kernel layer is responsible for maintaining the actual OPL stack composed of protected in-memory sensitive structures which can be modified and serves as a communication base for all kernel modules.
In order to easily deploy OPL on Linux systems and ensure a wider hardware compatibility without requiring any specific development, the well-known libpcap library is used to capture and redirect network traffic to kernel modules, instead of directly accessing resources through the hardware specific Ethernet driver, which is the Broadcom BCM2711 Ethernet driver for Rasperry Pi 4 platform.
This additional intermediate abstraction layer in the most resource critical layer implies non-negligible impacts on performance, especially with a general-purpose operating system only rendered real-time capable through PREEMPT-RT [13] patch, and even more on a quite resource limited Single Board Computer like the Raspberry Pi. For example, a cluster of fifteen Raspberry Pi 4-based OPL nodes, can easily exchange tens of bytes on a cyclic period of 10 ms, but any attempt to exchange large amount of non-critical data like video streaming during asynchronous period, will result on frequent cycle time overruns. This should theoretically not be possible but it's caused by accumulation of successive response times in the call tree and an overload of network resources consumption. The results are worse within a cluster of Raspberry Pi Zero nodes as isochronous-only exchange can also lead to cycle time overruns. In order to overcome these limitations, the PCAP layer must be removed and a direct communication established between kernel modules and Ethernet driver, as shown in Fig.5.

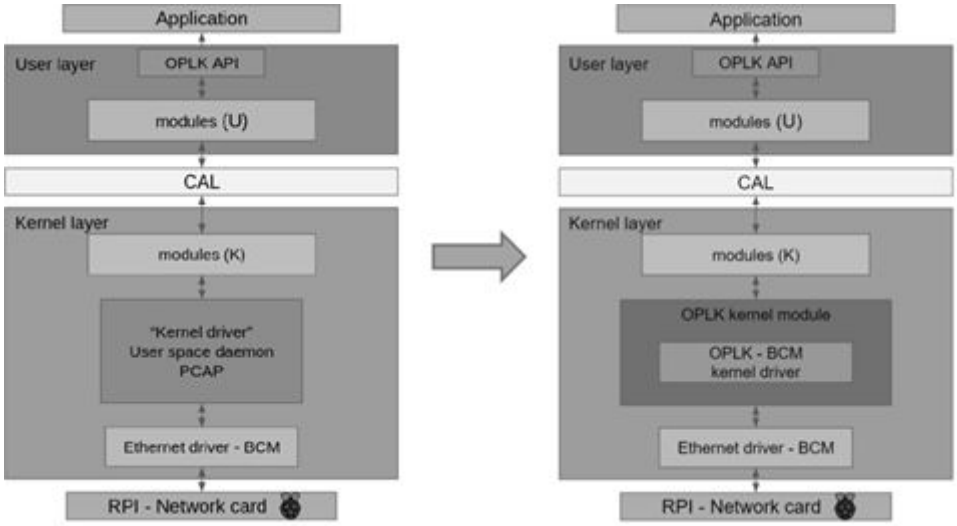

Figure 5: Initial and optimized OPL architectures

In practice, for the Raspberry Pi 4, a new OPL kernel module is implemented to encompass an intermediate dedicated driver that enables an efficient communication with the BCM Ethernet driver. This intermediate driver is called by OPL modules through external functions that provide high-level network capabilities, like sendTxBuffer(), getMacAddr() or changeRxFilter(). To build a simple and lightweight driver, we rely on Linux's generic platform_driver which is a pseudo-bus capable of connecting devices on busses with minimal infrastructure. Then, internal functions are in charge of interacting with two important structures, e.g. edrv and device driver, used as a shared data-set with the genuine Ethernet driver as shown in Fig.6. Unfortunately, the Raspberry Pi Zero does not have natively an Ethernet chipset, so an intermediary converter to a supported protocol as I2C, SPI or USB should be used. For a complete Precision Time Protocol (PTP) functionality Microchip's LAN9250 could be used, a simpler controller with SPI interface like ENC28J60 or ENC424J600 is also an option, whereas GPIO signal pins can be spared with an USB-based controller like the LAN9500A. This additional chipset may seem less straightforward compared to RPi 4's chipset, but these converters are fully documented with comprehensive datasheets and therefore can easily be integrated, which is not the case for the BCM chipset with only Linux's driver raw code as source of information. Regardless of the controller chosen, a similar intermediate driver must be implemented using the appropriate protocol-specific structure, for example usb_driver to achieve communication between OPL modules and LAN9500A USB-to-Ethernet controller.

In a typical general-purpose use case, the time needed to implement a driver for each of the chipset to be used with OpenPOWERLINK protocol may seem cumbersome, especially if we consider that each driver must evolve in line with operating system updates. But it's all the contrary from critical embedded systems point of view, as operating system updates are rare and integrated chipset have lifetimes that frequently exceed fifteen years. For example, we are running the last official PREEMPT-RT version for RPi e.g. rpi-4.19.y-rt which is three years old, and the most recent of the previously identified controllers was released in 2015.

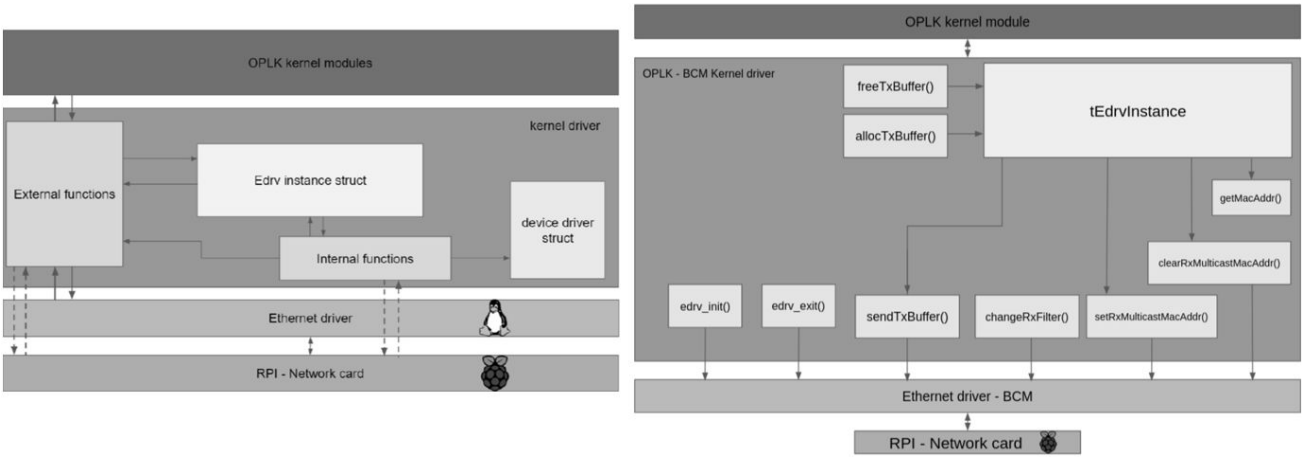

Figure 6: Overall and external functions-specific diagrams

3.3 Clock synchronization protocol

In order to precisely monitor each element of a launch system, in particular for safety and recovery purposes, analyze each mission's phase and related flight dynamics, or simply ensure a coherent actuators control within GNC's logic, it is essential to compute a timestamp for every gathered sensor and actuator data, and maintain a synchronicity of these timestamps at a higher system level. For example, Ariane 6 launch system must keep every time stamps precise within 10 µs in relation to other time stamps [6]. The old-fashion way to get such a system was to obtain Coordinated Universal Time (UTC) through a ground-based GPS receiver designed for precision timing, commonly equipped with an IRIG timecode [14] output. IRIG timecode are standard formats for transferring timing information via a continuous stream of binary data. The individual time code formats can be distinguished from one another by the signal characteristics, e.g. modulated versus unmodulated, which require different ways of signal transmission, by the data rate, and by the kind of information included in the transmitted data. For example, IRIG B0x2 or B1x2 refer to a 100Hz bit rate transmitting binary-coded decimal (BCD) day of year, hours and minutes, respectively through DC level shift or sine wave carrier. A dedicated on-board module could receive this information and use a fieldbus multipoint serial communication like RS-485 to widely exchange this master clock among every sub-systems.
However, in order to achieve an Ethernet-only avionics architecture, an Ethernet-based protocol must be used to enable a synchronized clock among the network nodes. The Network Time Protocol (NTP) is the most widely used clock synchronization protocol on the Internet. Still, it can achieve no more than one millisecond accuracy under ideal conditions which is not adequate to reach the necessary quality of synchronization for critical measurement and control systems. Fortunately, IEEE 1588v2 Precision Time Protocol [15] (PTP) specifically targets this issue of high accuracy synchronization and can achieve sub-microsecond accuracy with adequate hardware implementation. It's based on a hierarchical master-slave architecture for clock distribution, with an elected root timing reference called the grandmaster. The grandmaster transmits synchronization information to the clocks residing on its network segment. The distributed architecture of the system implies the occurrence of different rates of clock drift between the individual nodes. In order to avoid the individual local clocks to drift apart from each other, a clock readjustment mechanism is cyclically operated between the grandmaster and each slave nodes, as shown in Fig.7.

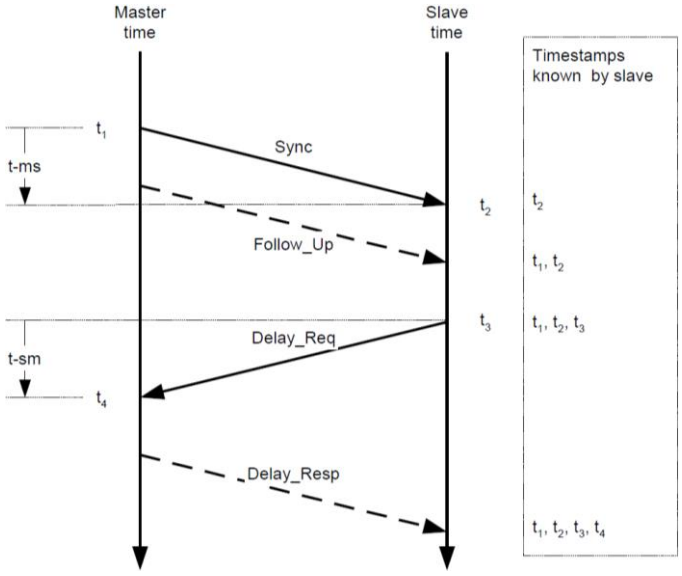

Figure 7: Basic synchronization message exchange [15]

The efficiency of this mechanism can be greatly improved if the uncertainty introduced by switches and every intermediary equipment is overcome. It can be possible if an end-to-end hardware support is done for every equipment in the network, as each node will modify conveyed PTP messages according to his own clock. The delivery variability across the network is compensated as timestamps in the messages are continuously corrected while traversing each equipment, instead of a unique update after a lengthy back and forth bounce from master to slave.
In fact, Linux operating system natively supports both software and hardware PTP implementations as long as the Ethernet controller and driver are compatible with at least one implementation. Unfortunately, Raspberry Pi 4 only support software-based implementation after a kernel workaround and recompilation, due to lack of hardware PTP support by the Broadcom BCM54213PE PHY, but the Compute Module 4's new form factor is updated with a BCM54210PE that will enable hardware support with observed performance within +/-50ns, thanks to an ongoing kernel update [16] and upcoming branch merge. Regarding Raspberry Pi Zero platform, as previously stated, an

additional Ethernet controller with PTP support like the LAN9250 must be used, but so far we have not managed to identify an external controller with an official manufacturer-provided kernel driver supporting hardware PTP, thus, an existing low-level Ethernet driver will probably need to be updated and recompiled if native support in Linux kernel is not released in the upcoming months.
For an end-to-end hardware support, a PTP-compliant Ethernet switch must also be identified. Such products are usually bulky with disproportionate overweight, but some companies [17] focused on size, weight and power sensitive applications are now proposing miniaturized equipment of approximatively 5 x 5 x 2 cm and a maximal consumption of 1.2W, without neglecting harsh environments support through ruggedized systems operating from -40°C to + 105°C. Such improvements are made possible with the use of single-chip Ethernet solution with hardware time-stamping at all PHY-MAC interfaces and a high-resolution hardware PTP clock, like Microchip's VSC7512 unmanaged switch featuring Layer-2 forwarding with basic VLAN and QoS processing, or KSZ8567 fully integrated Layer-2 managed switch with seven 10/100 Ethernet ports.

4. ASTREOS new-generation avionics

As a result of the followed Agile philosophy, a modular, cost-efficient and upgradeable avionics architecture is ready to serve as the basis for ASTREOS launchers, as shown in Fig.8, notably through the use of an **homemade generic extension board coupled to a mainstream SBC and a fully Ethernet-based communication network**.

One critical mission of the avionics is to **continuously provide a complete report on the behavior of the propulsion system** composed of three tanks containing azote, ethanol and liquid oxygen, in addition to several electro-valves in order to manage the supply of propellants to the engine. The status on propulsion system operationality must be computed through a variety of pressure, differential pressure and temperature sensors dispatched along the tanks, the supply lines and the engine. These measurement and control systems are represented as purple boxes on the diagram, and we can see that only three distributed systems on the upper, the middle and the lower part of the rocket are necessary to operate twenty five sensors and actuators. Moreover, thanks to a versatile extension board coupled with a Raspberry Pi Zero, these systems are easily interchangeable as specific electronic blocks have been conceived to ensure the support of all necessary digital and analog communication protocols, and all these blocks are integrated into a **unique extension board design**.
Another important goal for each mission is to **record the flight environment** especially during important phases like subsonic to trans- and hypersonic, notably to compare obtained pressures, global and local surface strains and level of vibrations with those expected through the simulation activities. These systems are spread out from the rocket nose to its base. They are identified by green boxes on the overview figure and share the same extension board design.

In order to facilitate the integration of more complex and resource-intensive flight-ready systems like the On Board Computer, a Video Module or the Inertial Measurement Unit, identified as dark blue boxes on the figure and dispatched on the upper avionics bay, we have designed **another extension board to match Raspberry Pi 4 form factor with reduced Analog-to-Digital capabilities**. In line with the strategy followed for acquisition and control systems, these complex systems can efficiently be switched from one position to another on the avionics bay, as they mostly rely on software implementation and share a common hardware platform.

One of the strongest requirements imposed by Esrange launch base, for recovery and safety issues, is the ability to **measure, compute and communicate the rocket trajectory**. For a reliable and precise trajectory during all propulsion, ballistic and return phases, a **data-fusion** must be performed of absolute positions which can be obtained via GNSS for example, and relative positions usually through Inertial Measurement Unit. So several antennas and receivers including GNSS, can naturally be observed on the synoptic as light blue boxes. In order to communicate trajectory, but also propulsion system status, video stream and any significant flight-related data to the launch base a circular S-band patch antenna and an S-Band transceiver is used for telemetry.
After a thorough survey, we have identified the **XLink-S transceiver** [18] from IQ Spacecom as a perfect match, since it only weighs 200 grams and is very compact i.e. easily fits into 1U CubeSat, and most importantly it can deliver a downlink data rate of up to 100 Mbps through an Ethernet interface and a Nano-D-Sub connector, which is completely in line with our fully Ethernet-based architecture. In order to limit the impact on performance caused by the multiplication of intermediate Ethernet devices, the number of switches is reduced to a strict minimum of 2, displayed as brown boxes, and do not exceed the limit of 3 switches in cascaded topology beyond which the performance is degraded. To meet future needs for additional systems, a mesh architecture will be implemented to maintain an optimal level of network quality.

Such **Ethernet architecture** is naturally **extended to the entire ground segment**, thus guaranteeing a strong compatibility between on-board and on-ground communication networks. For example, as tens of milliseconds

accuracy is sufficient for ground operations, NTP is used to synchronize all ground equipment based on an UTC forwarded by a dedicated GPS receiver. A strong synchronization between ground and on-board systems can then easily be put into action, as long as an on-board module executes an NTP client and assumes the role of master within the PTP avionics network. In fact, with the exception of some discrete signals used for instance to detect umbilical tear-off, all the communication between on-board and ground segment is done through **two pairs of twisted wires**, instead of the tens of wires necessary on previous PERSEUS's launchers.

This avionics architecture responds efficiently to the needs of the first iteration of ASTREOS, whose main goal is to allow to master the bi-liquid LOX-Ethanol propulsion as well as the solid propulsion used until now on PERSEUS. But as explained throughout this paper, thanks to a strategic highly adaptive, scalable-aware and modular-focused approach, with mostly **open-source software** and **COTS-based hardware** supported by a **generic mainstream platform** with minimal specific developments to achieve a flight-ready system, we will be able to quickly and incrementally integrate in a continuous Agile perspective, new building blocks to rapidly cover Thrust Vector Control (TVC), Autonomous Flight Termination System (AFTS), fin controls and other sophisticated GNC capabilities for ASTREOS upcoming iterations.

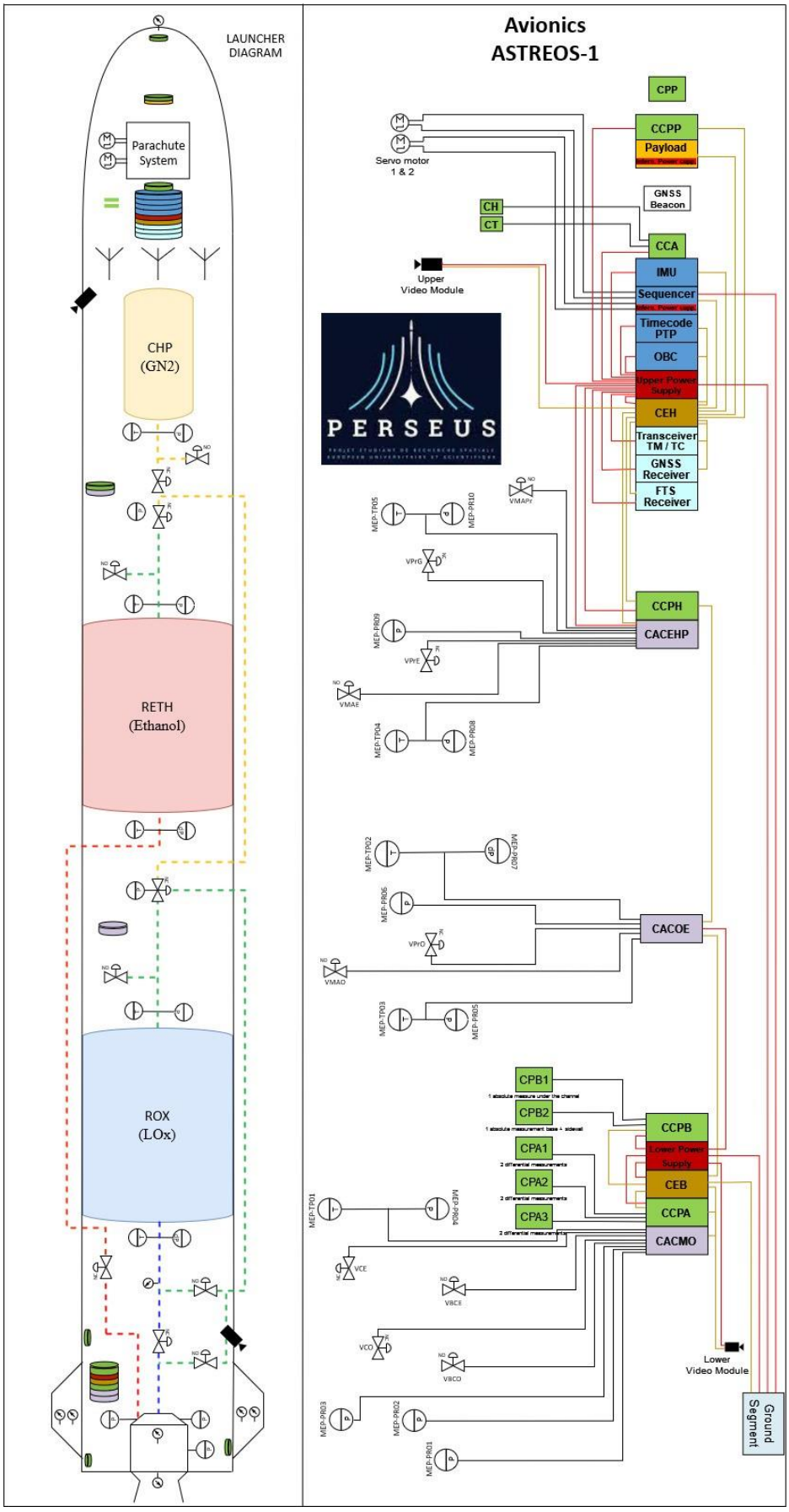

Figure 8: Avionics overview of the ASTREOS-1 rocket

Acknowledgements

Acknowledgements to the hundreds of students who invest themselves with passion and devotion to make this project grow, and without whom we would be unable to achieve these ever more ambitious objectives.
There is no doubt that the space domain of tomorrow will be built by brilliant engineers with the will to take humanity to the edge of space.

The authors would like to acknowledge the CNES for financial and technical support to PERSEUS avionics.

Special thanks to the PERSEUS trainees and the following student associations:
- GALLAND Jérémy who developed the telemetry system from scratch
- DUBREUCQ Thomas, RIBEIRO-MENDES Sébastiao and LALLIER Julien for building the premise of the full Ethernet architecture
- MERIEL Camille for her work on the NTP and PTP synchronization protocols
- SANTOS Vignon Gisèle for her contribution to the GNC
- CHAUVET Antoine and DEMEILLERS Thomas for their work on the extension card and the modular OBC
- TAAME Ilyasse for having reworked and updated the avionics synoptic
- the TopAero association of Sorbonne University for its work on the OpenPowerlink driver
- the members of the EUROAVIA association of the EPF for their strong involvement on the Flight Termination System

References

[1] PERSEUS : Le Projet Etudiant de Recherche Spatiale Européen. n.d. PERSEUS. https://www.perseusproject.com.
[2] Santos, Jose Maria Delos. 2021. What Is the Agile Manifesto? | Top Agile Principles & Values 2022. *Project-Management.Com* (blog). September 2, 2021. https://project-management.com/agile-manifesto/.
[3] Scrum (Software Development). 2022. In Wikipedia. https://en.wikipedia.org/w/index.php?title=Scrum_(software_development)&oldid=1089926422.
[4] Decotignie, J.-D. 2005. Ethernet-Based Real-Time and Industrial Communications. *Proceedings of the IEEE* 93 (6): 1102-17.
[5] Felser, M. 2005. Real-Time Ethernet - Industry Prospective. *Proceedings of the IEEE* 93 (6): 1118-29.
[6] Schalk, Kevin, Kai Müller, Karsten Peter, and Johannes Sebald. 2017. Microsecond-Precision Time Stamping in a Deterministic Distributed Sensor Network Utilizing OpenPOWERLINK. In *2017 IEEE International Conference on Wireless for Space and Extreme Environments (WiSEE)*, 52–56.
[7] Feng, Tieshan, Zhiyao Zhang, Feng Gao, and Jianlai Wang. 2020. Research of Real-Time Deterministic Ethernet in Launch Vehicle. In *2020 IEEE 5th Information Technology and Mechatronics Engineering Conference (ITOEC)*, 995-99.
[8] Felser, Max. 2010. Real Time Ethernet: Standardization and implementations. In *2010 IEEE International Symposium on Industrial Electronics*, 3766-71.
[9] Chongyuan, Hou, Jiang Hanhong, Yang Yuan, Rui Wanzhi, and Hu Lianwu. 2010. Research on Implementing Real Time Ethernet for Ship Power System.
[10] Robert, Jérémy, Jean-Philippe Georges, Eric Rondeau, and Thierry Divoux. 2010. Analyse de performances de protocoles temps-réel basés sur Ethernet. *Sixième Conférence Internationale Francophone d'Automatique, CIFA 2010*.
[11] Seno, L., S. Vitturi, and C. Zunino. 2009. Real Time Ethernet networks evaluation using performance indicators. In *2009 IEEE Conference on Emerging Technologies & Factory Automation*, 1-8.
[12] OpenPOWERLINK. https://sourceforge.net/projects/openpowerlink/.
[13] Realtime:Start [Wiki]. https://wiki.linuxfoundation.org/realtime/start
[14] 200 IRIG Serial Time Code Formats - Public RCC - TRMC Website (Confluence). https://www.trmc.osd.mil/wiki/display/publicRCC/200+IRIG+Serial+Time+Code+Formats.
[15] IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems. 2008. *IEEE Std 1588-2008 (Revision of IEEE Std 1588-2002)*, July, 1–269.
[16] CM4 Is Missing IEEE1588-2008 Support through BCM54210PE · Issue #4151 · Raspberrypi/Linux. n.d. GitHub. https://github.com/raspberrypi/linux/issues/4151.
[17] BotBlox. n.d. GitHub. https://github.com/botblox.
[18] XLink - S (SDR). https://www.iq-spacecom.com/products/xlink-s.